CRITERIA FOR SOFTWARE MODULARIZATION

David N. Card* - Gerald T. Page* - Frank E. McGarry**

*Computer Sciences Corporation, Silver Spring, MD 20910
**National Aeronautics and Space Administration, Greenbelt, MD 20771

ABSTRACT

A central issue in programming practice involves determining the appropriate size and information content of a software module. This study attempted to determine the effectiveness of two widely used criteria for software modularization, strength and size, in reducing fault rate and development cost. Data from 453 FORTRAN modules developed by professional programmers were analyzed. The results indicated that module strength is a good criterion with respect to fault rate, whereas arbitrary module size limitations inhibit programmer productivity. This analysis is a first step toward defining empirically based standards for software modularization.

INTRODUCTION

The module is the basic unit of software development, maintenance, and management. A basic activity of the software design process is the partitioning of the software specification into a number of program modules that together satisfy the original problem statement. To do this, programmers need criteria for defining the information content and organization of modules.

The major theoretical criteria for software modularization include strength/cohesion and coupling[1] and information hiding.[2] These criteria are, however, difficult to quantify. An independent observer of the development process cannot easily determine the levels of strength, coupling, and information hiding achieved in any given module. The use of these concepts is thus limited in an environment where quality assurance (as adherence to standards) is stressed.

Measures of size (number of source lines of code or executable statements) have consequently been adopted as a simple expedient.[3] Although many benefits have been claimed for module size limitations, at present there is no theoretical basis or empirical evidence for using module size as a criterion for software modularization.[4]

The purpose of this study was to compare the effectiveness of size (e.g., a 60-line-module standard) and a theoretically based measure (strength) as criteria for software modularization. Strength (or singleness of purpose) was chosen for this comparison because, like size, it can be determined from the contents of a single module. Measuring coupling or information hiding requires that more than one module at a time be examined.

This study, therefore, compares the effectiveness of module strength and size criteria with respect to module cost and fault rate. Although maintainability (or modifiability) is another important software attribute, it was not possible to measure or analyze it in this study. Because some programmers generally produce low-fault, low-cost modules while others produce expensive, faultprone modules, it was also necessary to investigate the interaction of these criteria with individual programmer performance.

DATA ANALYZED

This study examines data from 453 new FORTRAN modules developed by 26 professional programmers for 5 major software development projects. The term "module" has been defined in many different ways. For the purposes of this study, it refers to a FORTRAN subroutine, or the smallest program unit that is independently compilable. Although more sophisticated languages are available, many organizations rely on FORTRAN for scientific computing applications. This study is thus relevant to current practice. Furthermore, these modularization criteria seem likely to remain important considerations in software development using new languages such as Ada† (for which extensive data are not yet available).

The Software Engineering Laboratory[5] (SEL) collected these data as part of an ongoing program of software measurement and technology evaluation. The SEL is a research project sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) and supported by Computer Sciences Corporation and the University of Maryland. The SEL

†Ada is a registered trademark of the U.S. Government, Ada Joint Program Office.

studies software developed for spacecraft flight dynamics applications. These systems provide ground-based support for spacecraft navigation and control. Typical projects produce from 30,000 to 150,000 source lines of code.

Module Strength

Myers[6] defines seven levels of module strength. In descending order, these are functional, informational, communicational, procedural, classical, logical, and coincidental. A high (functional)-strength module performs a single well-defined function. Myers contends that high-strength modules are superior to low-strength modules. Although it was not possible to test this theory exactly, a reasonable approximation was made. Although some recent attempts to develop objective measures of module strength[7,8] seem promising, they are not (in their present forms) easily applied. Consequently, they were not employed in this study.

Instead, programmers determined the strength of a module using a checklist. Programmers rated each module they developed as performing one or more of the following functions: input/output, logic/control, and algorithmic processing. Distinguishing the types of functions seemed to be a less ambiguous task than identifying the number of functions, because the number of functions depends on the level of decomposition recognized by the respondent. Performing a single function type is a necessary (but not sufficient) condition for high module strength.

Those modules described as having only one function were classified as high strength; those described as having two functions were classified as medium strength; and those modules described as having three or more functions rated low strength. Table 1 summarizes the results of this classification process.

Table 1. Module Strength Distribution

MODULE STRENGTH	NUMBER OF FORTRAN MODULES	MEAN EXECUTABLE STATEMENTS	MEAN DECISIONS PER EXECUTABLE STATEMENT
LOW	90	77	0.29
MEDIUM	176	60	0.32
HIGH	187	48	0.32

Module Size

The 453 modules in the sample were classified into three approximately equal ordered groups on the basis of the number of executable statements in each module. Table 2 shows the results of this classification.

The largest module in the sample contained 267 executable statements. The dividing line of 31 executable statements is significant because, in the environment studied, it corresponds to about 60 source lines of code. Many programming standards[3] limit module size to one page (or 50 to 60 source lines of code). The informal guideline used in this environment is that no module should exceed 2 pages (about 64 executable statements). Military standards on module size range from 50 to 200 executable statements.[4] One purpose of the study was to test the validity of such standards, in general, and, in particular, to determine if the local guideline should be strengthened.

Table 2. Module Size Distribution

MODULE SIZE	NUMBER OF FORTRAN MODULES	EXECUTABLE STATEMENTS	MEAN DECISIONS PER EXECUTABLE STATEMENT
SMALL	154	1 TO 31	0.31
MEDIUM	148	32 TO 64	0.31
LARGE	151	65 OR MORE	0.32

ANALYSIS RESULTS

The objective of the analysis was to determine the effect of module size and strength criteria on quality measures, that is, the module cost (number of hours per executable statement) and fault rate (number of faults per executable statement). An initial examination of the data revealed that neither module cost nor fault rate was normally distributed. Figures 1 and 2 illustrate these phenomena. Consequently, the authors adopted contingency table and nonparametric correlation approaches to the analysis rather than relying on normal-distribution-based techniques such as regression and analysis of variance.

To perform the contingency table analysis, every module was assigned to one of three ordered classes (of nearly equal size) for each of the quality measures of cost (low, medium, high) and fault rate (zero, medium, high). The values 0.151 and 0.322 programmer hour per executable statement divided the modules into the three cost classes (i.e., 0.151 or less was low cost). Faults were counted for each module from the completion of unit testing until the end of acceptance testing. The value 0.045 fault per executable statement distinguished between medium- and high-fault-rate classes. One class consisted of those modules with no faults. It was thus possible to form a series of 3-by-3 tables, each comparing classes of module strength or size with classes of module cost or fault rate.

The strength of relationships was assessed by calculating the gamma (γ) correlation statistic[9] between the ordered classes of modularization criteria and quality measures. This statistic varies from -1.0 to +1.0. For example, a perfect negative correlation (-1.0) would result only if all high-strength modules had zero faults, all medium-strength modules had medium fault rates, and all low-strength modules had high fault rates. Variations in programmer performance also affect module cost and fault rate[10]; therefore, this factor was also considered in the general analysis as well as in a subsequent analysis.

General Results

Initially, module strength and size were cross-tabulated with cost and fault rate. Lines 1 and 4 of Table 3 list the correlation coefficients obtained from this analysis. Significant relationships were found between module strength and fault rate ($\gamma = -0.35$) and between module size and cost ($\gamma = -0.31$). The criterion for significance (probability of error less than 0.001) is very conservative. These correlations seem low, but Figures 3 and 4 provide better illustrations of the magnitude of these relationships. Fully 50 percent of high-strength modules were fault-free while only 18 percent of low-strength modules were fault-free. Similarly, 46 percent of large modules fell into the lowest cost class, whereas just 22 percent of the small modules were rated as low cost.

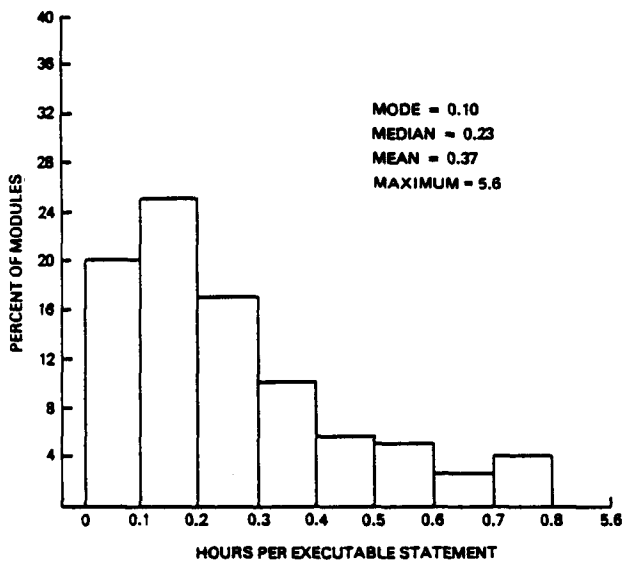

Figure 1. Distribution of Cost

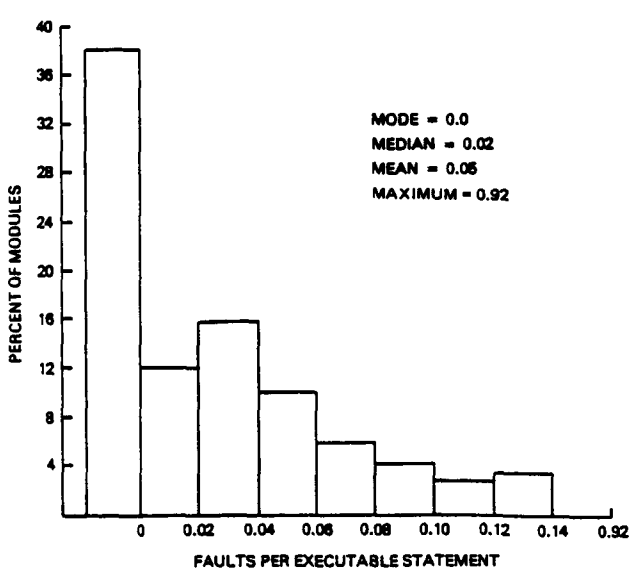

Figure 2. Distribution of Faults

Table 3. Contingency Table Results

CRITERIA	EFFECT CONTROLLED	CORRELATIONS[a]		LINE
		FAULT RATE	COST RATE	
MODULE STRENGTH	NONE	-0.35[b]	-0.19	1
	SIZE	-0.32[b]	-0.27[b]	2
	PROGRAMMER	-0.21	0.10	3
MODULE SIZE	NONE	0.20	-0.31[b]	4
	STRENGTH	0.19	-0.38[b]	5
	PROGRAMMER	0.27[b]	-0.41[b]	6

[a]GAMMA (γ) STATISTIC.
[b]PROBABLY LESS THAN 0.001 THAT CORRELATION IS ACTUALLY ZERO.

Table 1 indicates, however, that module strength and size might be related to each other. Low-strength modules tend to be larger. Lines 2 and 5 of Table 3 show the (partial) correlations obtained for module strength and size individually while controlling (removing) the effect of the other. The relationships with module fault rate remain essentially unchanged. There is, however, some interaction between module strength and size with respect to module cost. (Compare line 1 versus line 2 and line 4 versus line 5 in Table 3.)

Controlling for module size, the correlation between module strength and cost increases from -0.19 to -0.27 and becomes significant. Controlling for module strength, the correlation between module size and cost increases from -0.31 to -0.38. These results imply that, overall, high-strength modules (usually small) tend to be low cost but that large modules also tend to be low cost (independent of module strength). Another study[11] identified a similar relationship between module size and cost for a very different type of software.

One previous study[12] that found a lower fault rate for larger modules based its conclusions on the behavior exhibited by a small sample of large modules. Another study[10] applied parametric regression to a larger sample from the same data base as this study. As discussed earlier, that statistical approach is inappropriate for non-normally distributed data. Although these results contradict the two previous studies of fault rate, the current results appear to be more robust.

Thus far, the potential effects of programmer performance were ignored. Lines 3 and 6 of Table 3 show the correlations between the modularization criteria and quality measures obtained while controlling for the effect of programmer performance. (The interaction of module size and strength is, however, no longer controlled.) The large changes from the initial correlations demonstrate that programmer performance interacts with both module size and strength. The disappearance of the significance of the relationships between module strength and

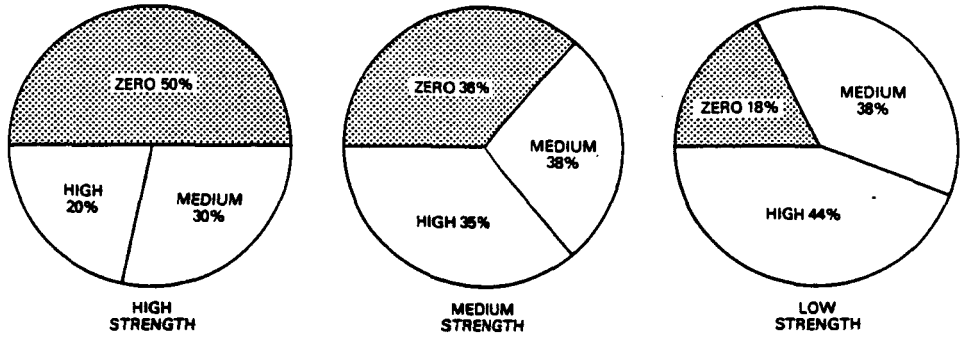

Figure 3. Fault Rate for Classes of Module Strength

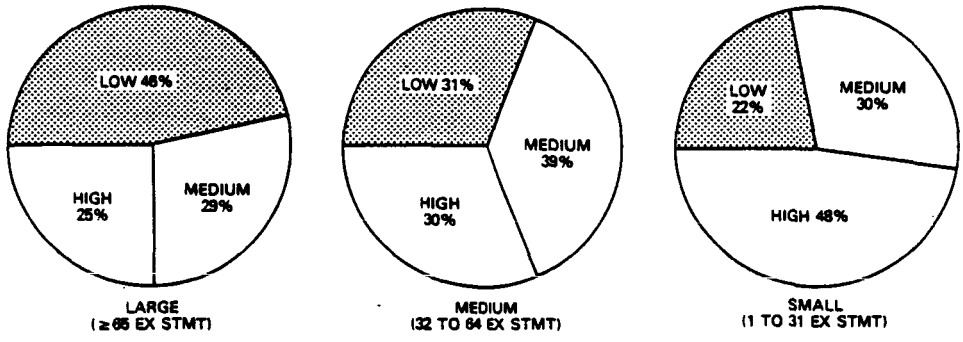

Figure 4. Development Cost for Classes of Module Size

module cost and fault rate indicates that these relationships exist because high-strength modules are associated with programmers who produce modules that cost less and have low module fault rates.

Programmer-Specific Results

The effect of programmer performance was also examined in a subsequent analysis. Of the 26 programmers in the sample, 16 developed 9 or more modules. Together these programmers accounted for 413 of the total 453 modules. The performance of these programmers was reanalyzed using nonparametric correlation[9] to better define the relationship of programmer performance to modularization criteria. Table 4 summarizes the data obtained from the 16 programmers.

For each of these programmers, the percent of zero-fault and low-cost modules was computed. Table 5 shows the correlations (by programmer) between the modularization criteria and the quality measures. Programmers who produce low-fault-rate modules (i.e., "good" programmers) tend to produce high-strength modules. Good programmers do not, however, appear to have any preference for a particular module size. The lower significance levels associated with the correlation coefficients result from the reduction in sample size produced by studying 16 programmers instead of 453 modules.

Table 4. Programmer Data Summary

PROGRAMMER	NUMBER OF FORTRAN MODULES	MEAN EXECUTABLE STATEMENTS	MEAN DECISIONS PER EXECUTABLE STATEMENT
A	46	46	0.30
B	25	45	0.35
C	25	57	0.40
D	28	80	0.33
E	9	53	0.23
F	54	51	0.35
G	24	62	0.32
H	30	71	0.31
I	18	77	0.29
J	50	56	0.26
K	17	47	0.41
L	40	48	0.31
M	13	64	0.39
N	9	90	0.33
O	16	38	0.30
P	9	53	0.34

Table 5. Nonparametric Correlation Results (by Programmer)

CRITERIA	CORRELATIONS[a]	
	FAULT RATE	COST RATE
MODULE STRENGTH	−0.53[b]	−0.29
MODULE SIZE	−0.17	−0.18

[a]SPEARMAN CORRELATION COEFFICIENT.

[b]PROBABILITY LESS THAN 0.05 THAT CORRELATION IS ACTUALLY ZERO.

Figure 5 illustrates the relationship between module strength and the fault rate. Although the trend is clear, a great deal of unexplained variation is also present. Good programming consists of more than just writing high-strength modules.

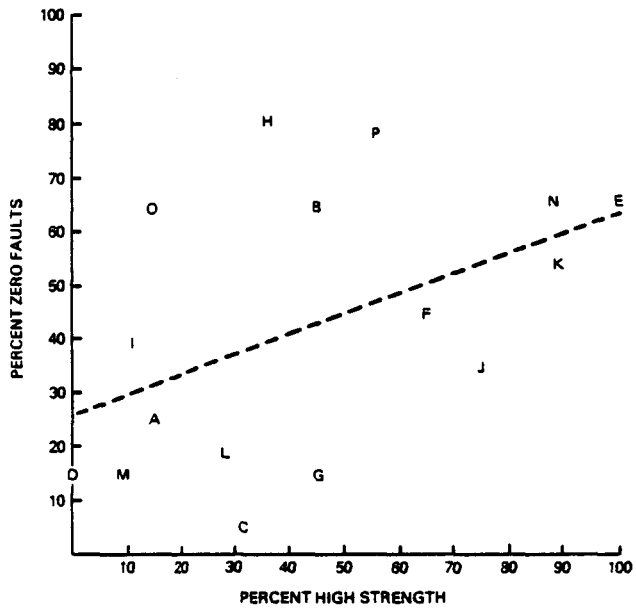

Figure 5. Module Strength and Faults by Programmer

CONCLUSIONS

The preceding discussion examined the relationship between modularization criteria and quality measures from two perspectives: their overall effect and the contribution of individual programmer performance. Conclusions based on the contingency table analysis (lines 2 and 5 of Table 3) are correct as stated. Finding that programmer performance accounts for some of the strength of these relationships does not affect their validity. However, this result does highlight the difficulty of separating the effects of programmer performance from those of technology or methodology.[13] Furthermore, it enables us to learn about software development in the way that Soloway[14] prescribes, by observing what good programmers do. Conclusions based on the preceding analysis are as follows:

- Good programmers tend to write high-strength modules.
- Good programmers show no preference for any specific module size.
- Overall, high-strength modules have a lower fault rate and cost less than low-strength modules.
- Overall, large modules cost less (per executable statement) than small modules.
- Fault rate is not directly related to module size.

These conclusions suggest that module size should not be arbitrarily limited by any programming standard. Two-thirds of the modules in this sample fell below the local size guideline of two pages (about 64 executable statements), even though this is not an enforced standard. As noted by Bowen[4], the application of a good design methodology usually results in modules well below the common size limits.

Generally, programmers should be encouraged to write high-strength modules but to make those modules large enough to encompass an entire function. Because low-strength modules are likely to be larger than average, a module size criteria may have an indirect favorable effect on the fault rate. However, the cost advantages associated with larger modules dictate that large, high-strength modules must also be acceptable. Large modules may be appropriate for some types of software (for example, mathematical algorithms).

Programmers, especially the less experienced ones, should be encouraged to write high-strength modules because this is a characteristic of successful programmers. The further development of objective measures of module strength may make this criterion more palatable to organizations that use formal quality assurance procedures. A better measure of module strength should show an even higher correlation with fault rate. In the interim, a simple checklist of the number of types of functions performed can provide a simple but effective assessment of strength for quality assurance purposes.

REFERENCES

[1] W. P. Stephens, G. J. Meyers, and L. L. Constantine, "Structured Design," IBM Systems Journal, 1974, vol. 13, no. 2, pp. 115-139

[2] D. L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules," ACM Communications, December 1972, vol. 15, no. 12, pp. 1053-1058

[3] B. W. Kernighan and P. S. Plauger, The Elements of Programming Style. New York: MCGraw Hill, 1974, p. 126

[4] J. D. Bowen, "Module Size: A Standard or Heuristic?," Journal of Systems and Software, 1984, no. 4, pp. 327-332

[5] D. N. Card, F. E. McGarry, G. T. Page, et al., The Software Engineering Laboratory, NASA/GSFC, February 1982

[6] G. J. Myers, Composite/Structured Design. New York: Van Nostrand Reinhold, 1978

[7] R. D. Cruickshank and J. E. Gaffney, "Measuring the Development Process: Software Design Coupling and Strength Matrices," Proceedings of the Fifth Annual Software Engineering Workshop, NASA/GSFC, November 1980

[8] T. J. Emerson, "A Discriminant Metric for Module Cohesion," Proceedings of the Seventh International Conference on Software Engineering, 1984, pp. 294-303

[9] L. A. Marascuilo and M. McSweeney, Nonparametric and Distribution Free Methods for the Social Sciences. California: Brooks/Cole, 1977, pp. 466-471, pp. 431-435

[10] F. E. McGarry, "Measuring Software Development Technology," Proceedings of the Seventh Annual Software Engineering Workshop, NASA/GSFC, December 1982

[11] P. C. Belford, R. C. Berg, and T. L. Hannan, "Central Flow Control Software Development: A Case Study of the Effectiveness of Software Engineering Techniques," Proceedings of the Fourth International Conference on Software Engineering, September 1979, pp. 85-93

[12] V. R. Basili and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," ACM Communications, January 1984, vol. 27, no. 1, pp. 42-52

[13] D. N. Card, F. E. McGarry, and G. T. Page, "Evaluating Software Engineering Technologies," Proceedings of the Eighth Annual Software Engineering Workshop, NASA/GSFC, November 1983

[14] D. Littman, K. Ehrlich, E. Soloway, and J. Black, "You Can Observe a Lot by Just Watching How Designers Design," Proceedings of the Eighth Annual Software Engineering Workshop, NASA/GSFC, November 1983